Preparing Fusion Codes for Perlmutter

Igor Sfiligoi
San Diego Supercomputer Center
Under contract with General Atomics

SDSC SAN DIEGO SUPERCOMPUTER CENTER

NUG 2022

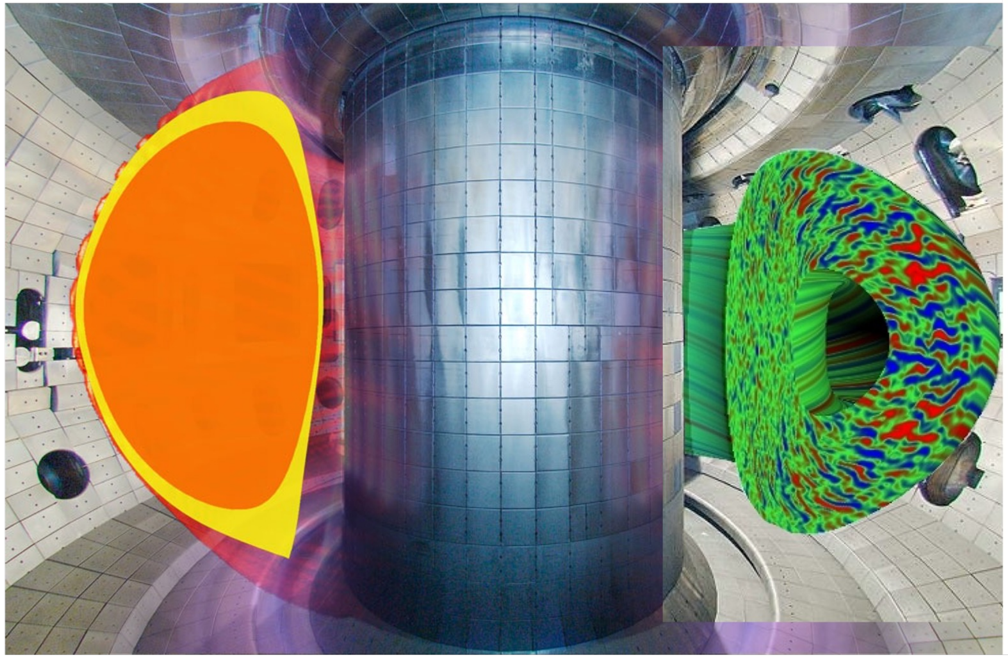
GENERAL ATOMICS

This talk focuses on CGYRO

- There are many tools used in Fusion research
- This talk focuses on **CGYRO**
  - An Eulerian fusion plasma turbulence simulation tool
  - Optimized for multi-scale simulations
  - Both memory and compute heavy

https://gafusion.github.io/doc/cgyro.html

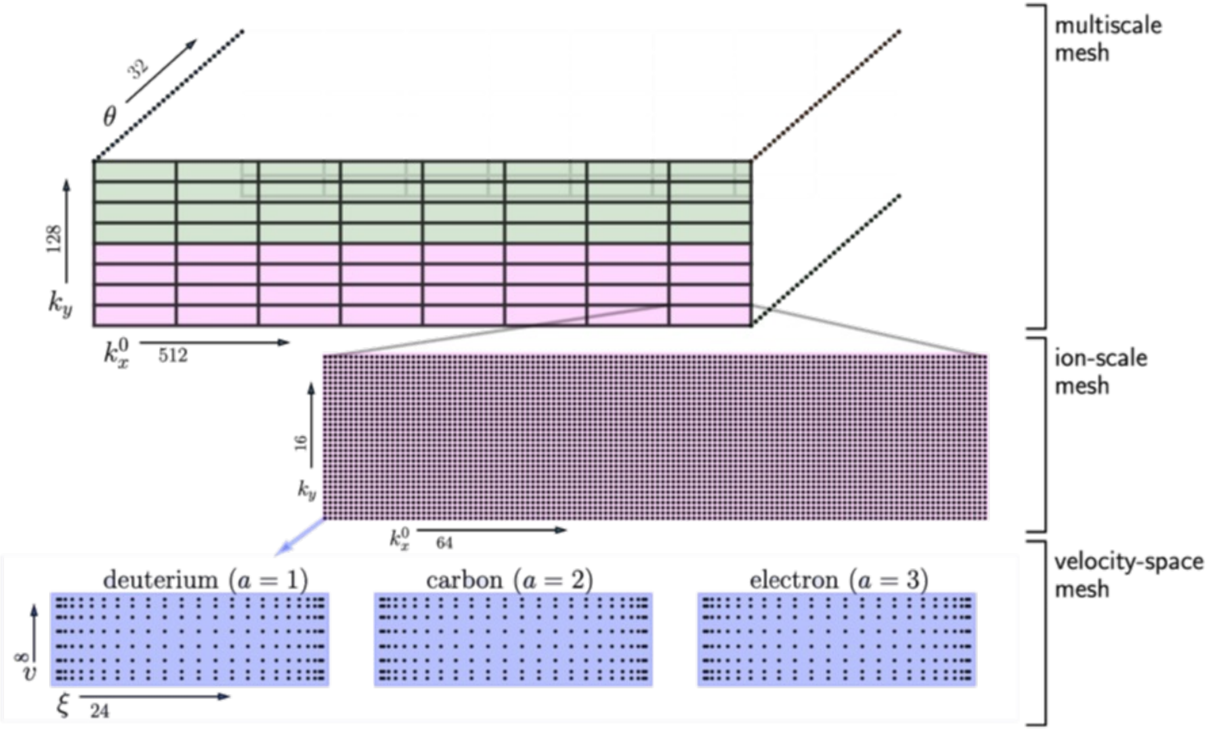
E. Belli and J. Candy
main authors GENERAL ATOMICS



Experimental methods are essential for gathering new operational modes. But simulations are used to validate basic theory, plan experiments, interpret results on present devices, and ultimately to design future devices.

CGYRO inherently parallel

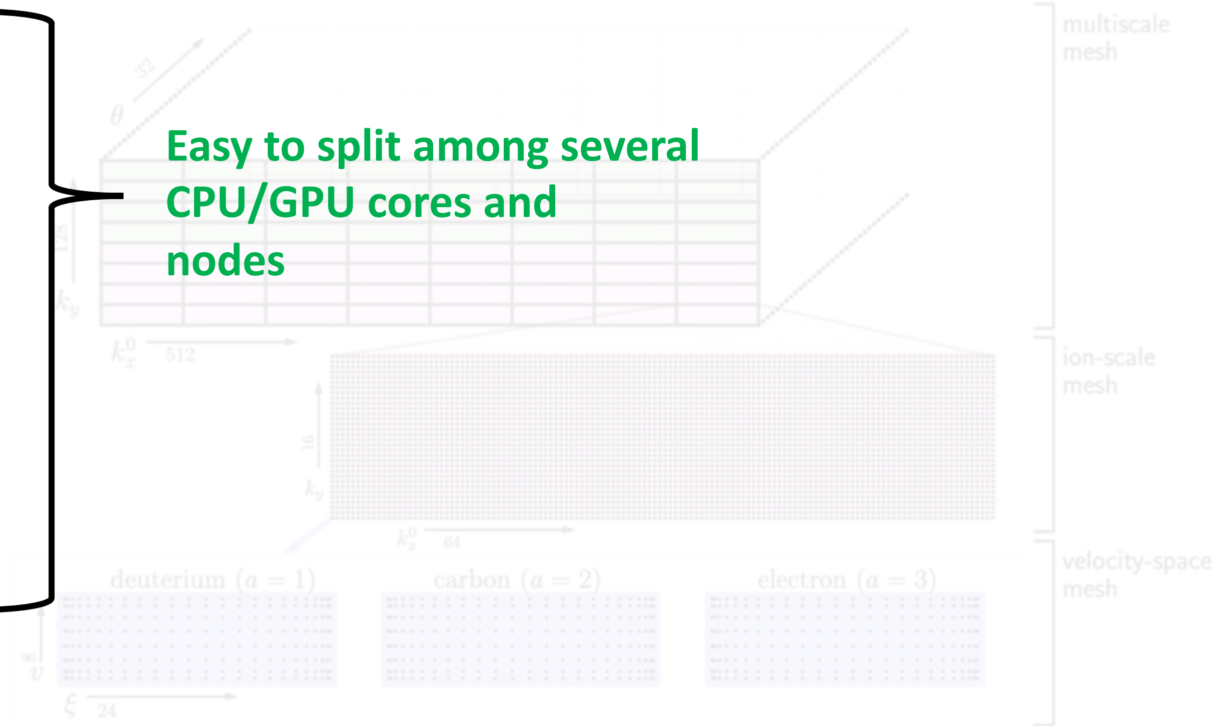
- Operates on 5+1 dimensional grid
- Several steps in the simulation loop, where each step
  - Can cleanly partition the problem in at least one dimension
  - All dimensions compute-parallel



CGYRO inherently parallel

- Operates on 5+1 dimensional grid
- Several steps in the simulation loop, where each step
  - Can cleanly partition the problem in at least one dimension
  - All dimensions compute-parallel

Easy to split among several CPU/GPU cores and nodes

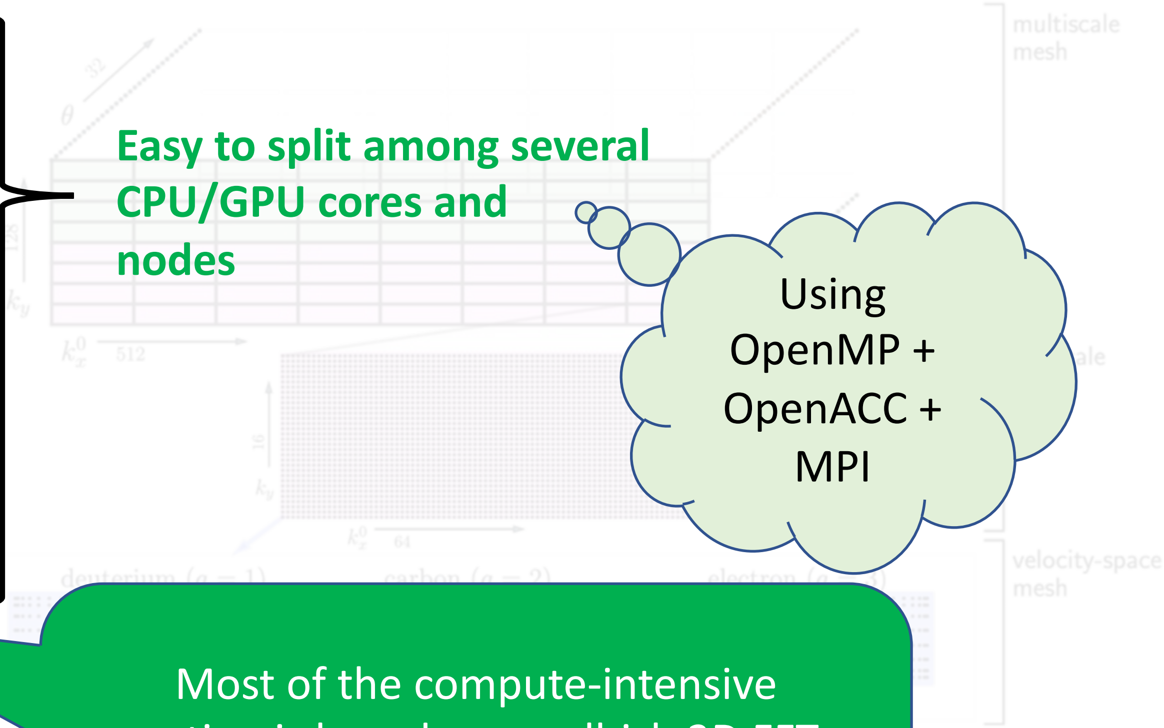


CGYRO inherently parallel

- Operates on 5+1 dimensional grid
- Several steps in the simulation loop, where each step
  - Can cleanly partition the problem in at least one dimension
  - All dimensions compute-parallel

Easy to split among several CPU/GPU cores and nodes

Using OpenMP + OpenACC + MPI



Most of the compute-intensive portion is based on small-ish 2D FFTs

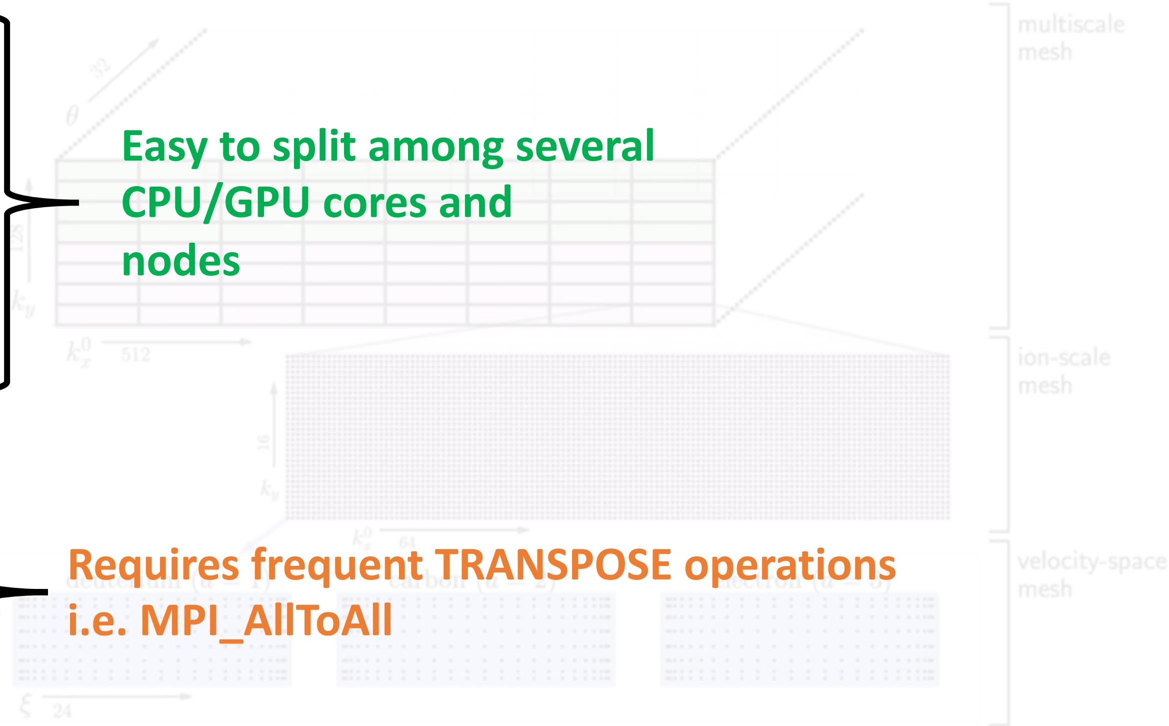
Can use system-optimized libraries

CGYRO inherently parallel

- Operates on 5+1 dimensional grid
- Several steps in the simulation loop, where each step
  - Can cleanly partition the problem in at least one dimension
    - But no one-dimension in common between all of them
  - All dimensions compute-parallel
    - But some dimension may rely on neighbor data from previous step

Easy to split among several CPU/GPU cores and nodes

Requires frequent TRANSPOSE operations
i.e. MPI_AllToAll

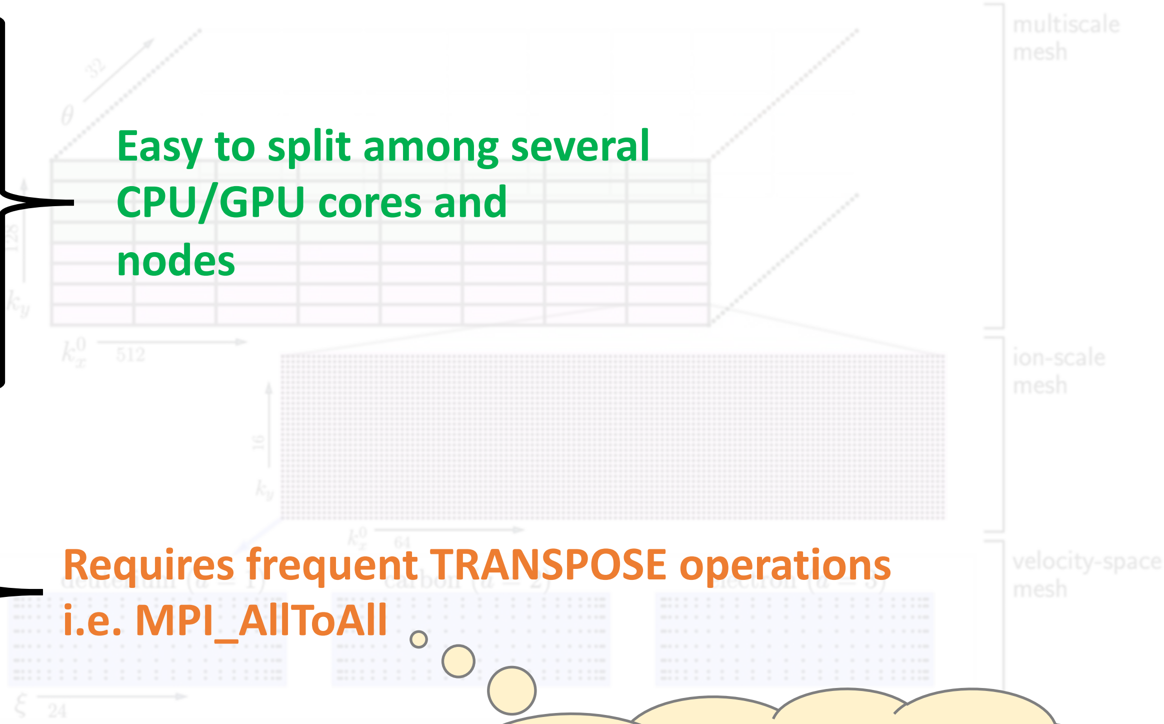


CGYRO inherently parallel

- Operates on 5+1 dimensional grid
- Several steps in the simulation loop, where each step
  - Can cleanly partition the problem in at least one dimension
    - But no one-dimension in common between all of them
  - All dimensions compute-parallel
    - But some dimension may rely on neighbor data from previous step

Easy to split among several CPU/GPU cores and nodes

Requires frequent TRANSPOSE operations i.e. MPI_AllToAll



Exploring alternatives but none ready yet

Cori and Perlmutter

- Cori was long a major CGYRO compute resource
  - And we were very happy with KNL CPUs
  - Lots of (slower) cores was always better than fewer marginally-faster cores
- CGYRO was ported to GPUs first for ORNL Titan
  - Then improved for ORNL Summit
    (Titan’s K80’s have severe limitations, like tiny memory and limited comm.)
- Deploying on Perlmutter (GPU partition) required just a recompilation
  - It just worked
  - Most of the time since spent
    on environment optimizations, e.g. NVIDIA MPS

Already had
experience with A100s
from Cloud compute

CPU vs GPU code paths

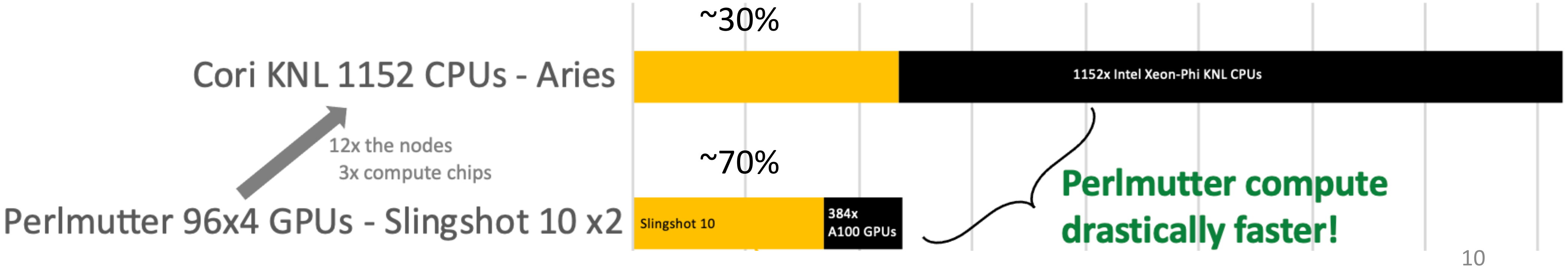
- CGYRO uses a OpenMP+OpenACC(+MPI) parallelization approach
  - Plus native FFT libraries, FFTW/MKL on Cori, cuFFT on Perlmutter
- Most code identical for the two
  - Enabling OpenMP or OpenACC based on compile flag
  - A few loops have specialized OpenMP vs OpenACC implementations (but most don't)
  - cuFFT required batch execution (reminder, many small FFTs)
- Efficient OpenACC requires careful memory handling
  - Especially when interacting with IO / diagnostics printouts
  - Was especially a problem while porting pieces of the code to GPU (now virtually all compute on GPU, partitioned memory just works)

Importance of great networking

- CGYRO communication intensive
  - Large memory footprint + frequent MPI_AllToAll
  - Non-negligible MPI_AllReduce, too
- First experience on Perlmutter with Slingshot 10 a mixed bag
  - Great compute speed
  - But simulation bottlenecked by communication

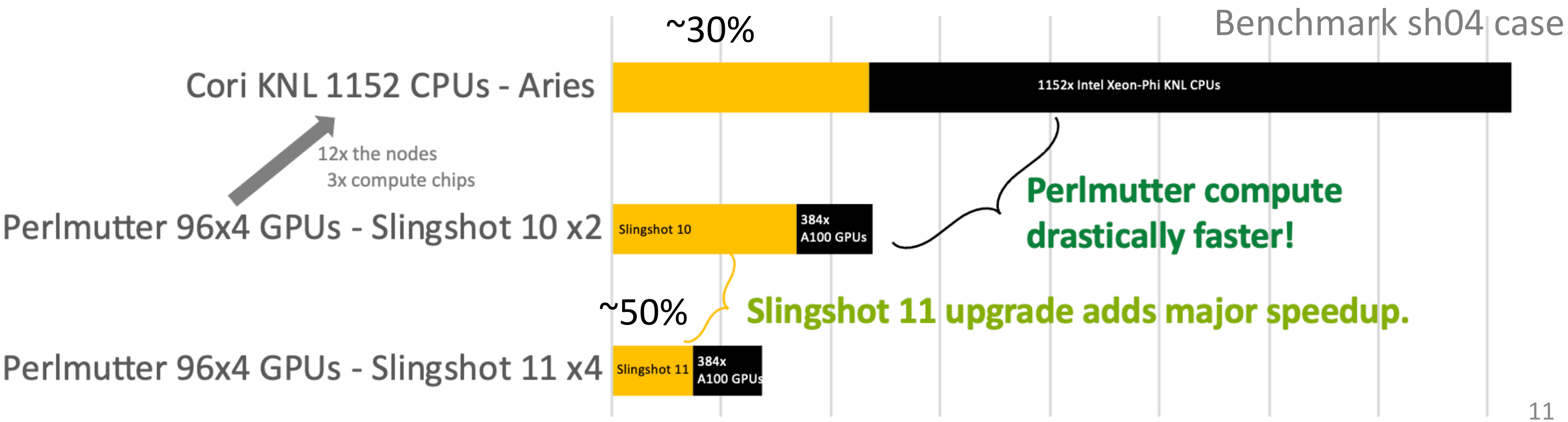
Benchmark sh04 case

Preview of SC22 poster



Importance of great networking

- CGYRO communication intensive
  - Large memory footprint + frequent MPI_AllToAll
  - Non-negligible MPI_AllReduce, too
- First experience on Perlmutter with Slingshot 10 a mixed bag
- The updated Slingshot 11 networking makes us much happier



Preview of SC22 poster

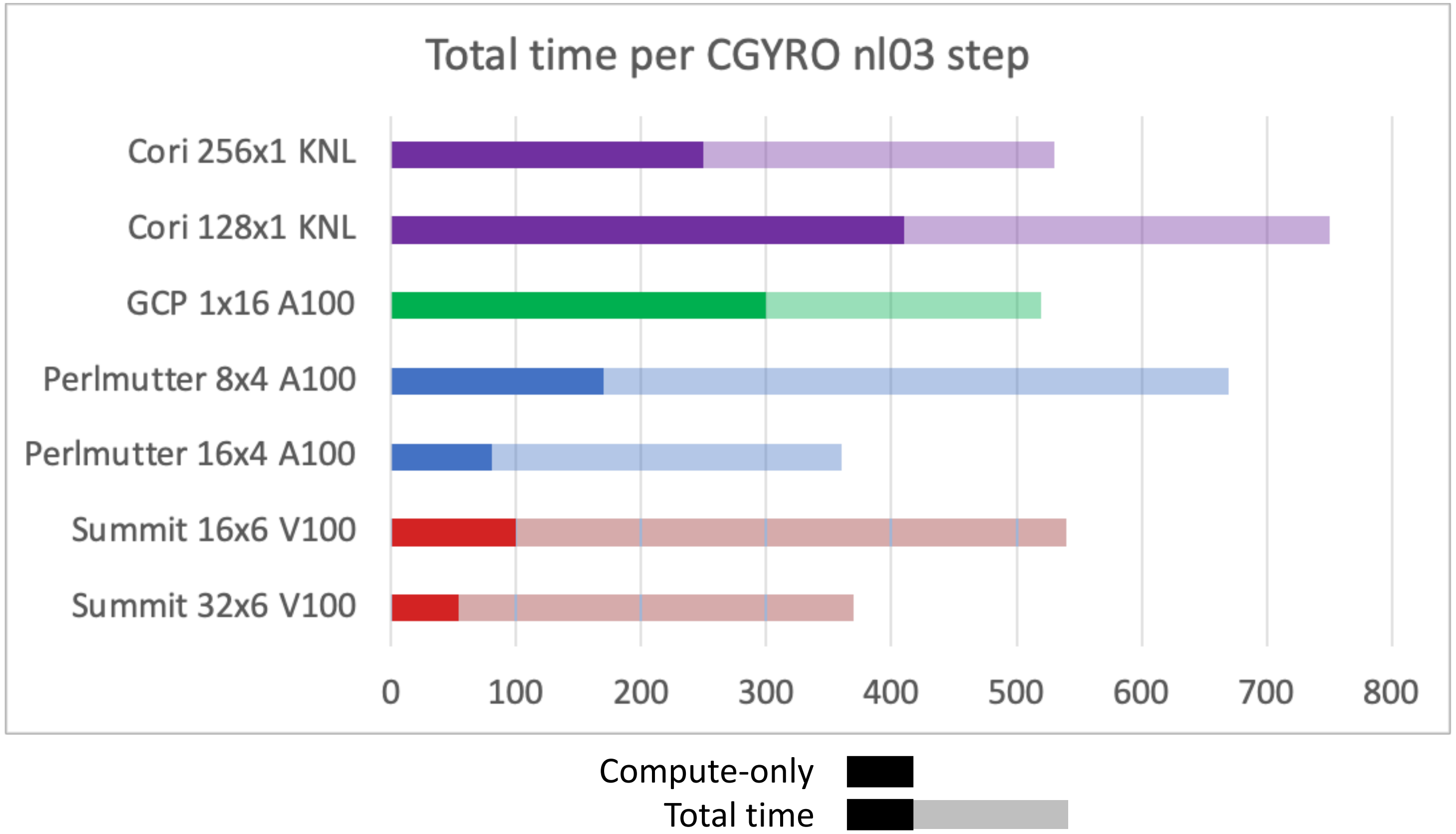
Importance of great networking

- CGYRO communication intensive
  - Large memory footprint + frequent MPI_AllToAll
  - Non-negligible MPI_AllReduce, too
- First experience on Perlmutter with Slingshot 10 a mixed bag
- The updated Slingshot 11 networking makes us much happier
  - But brings new problems
  - SS11 does not play well with MPS
    - Gets drastically slower when mapping multiple MPI processes per GPU
    - Something we are currently relying on for optimization reasons
  - Not a showstopper, but slows down our simulation in certain setups
    - NERSC ticket open, hopefully can be fixed
    - But we are also working on alternatives in CGYRO code

Disk IO light

- CGYRO does not have much disk IO
  - Updates results every O(10 mins)
  - Checkpoints every O(1h)
- Uses MPI-mediated parallel writes
  - Only a couple files, one per logical data type

A comparison to other systems



SS 10

Single GCP node faster than 16x Summit nodes

Looking at compute-only, Perlmutter’s A100s about twice as fast as Summit’s V100s

Presented at PEARC22 - https://doi.org/10.1145/3491418.3535130

Summary and Conclusions

- Fusion CGYRO users happy with transition from Cori to Perlmutter
  - Much faster at equivalent chip count
  - Porting required just a recompile
- Perlmutter still in deployment phase
  - Had periods when things were not working too great
  - But typically transient, hopefully will stabilize
  - Waiting for the quotas to be raised (128 nodes is **not** a lot for CGYRO)
- Only known remaining annoyance is SS11+MPS interference

Acknowledgements

- This work was partially supported by
  - The U.S. Department of Energy under awards DE-FG02-95ER54309, DE-FC02-06ER54873 (Edge Simulation Laboratory) and DE-SC0017992 (AToM SciDAC-4 project).
  - The US National Science Foundation (NSF) Grant OAC-1826967.
  - An award of computer time was provided by the INCITE program.
  - This research used resources of the Oak Ridge Leadership Computing Facility, which is an Office of Science User Facility supported under Contract DE-AC05-00OR22725.
  - Computing resources were also provided by the National Energy Research Scientific Computing Center, which is an Office of Science User Facility supported under Contract DE-AC02-05CH11231.